DATA STRUCTURES
USING "C"
(Single Link List)

For

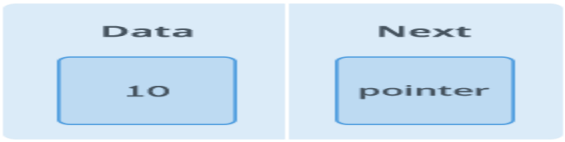
BCA Part-II (Session 2018-21) Students

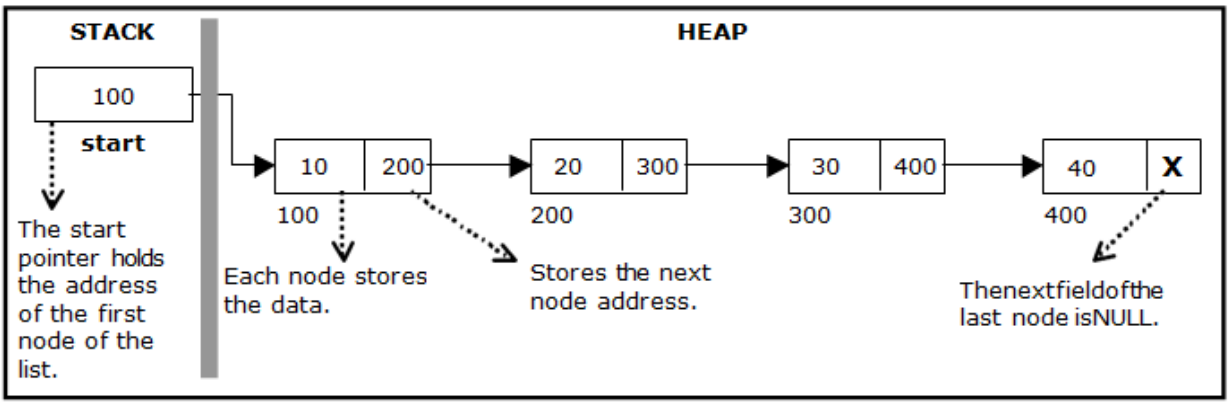
BY

ANANT KUMAR
MCA, M. Phil, M. Tech.
Faculty Member
Department of Computer Science
J. D. Women's College, Patna

Single Linked List

It is also called as Linear List or One Way List. It is linear collection of data elements called node. In this link list each node is divided into two parts. The first part contain information about an element and second part contain the address of the next node called *nextpointer* field.





Single Linked List

The basic operations in a single linked list are:

- Creation.
- Insertion.
- Deletion.
- Traversing.

```
# include <stdio.h>
# include <conio.h>
 # include <stdlib.h>

typedef  struct slinklist
{
     int data;
     struct slinklist *next;
} node;

node *start = NULL;

node* getnode();
int countnode(node *);
void createlist(int);
void traverse();
void rev_traverse(node *);
void insert_at_beg();
void insert_at_end();
void insert_at_mid();
void delete_at_beg();
void delete_at_last();
void delete_at_mid();

void main()
{
        int ch;
        while(1)
        {
                clrscr();
                printf("\n 1.Create a list \n");
                printf("\n 2.Insert a node at beginning ");
                printf("\n 3.Insert a node at end");
                printf("\n 4.Insert a node at middle\n");
                printf("\n 5.Delete a node from beginning");
                printf("\n 6.Delete a node from Last");
                printf("\n 7.Delete a node from Middle\n");
                printf("\n 8.Traverse the list (Left to Right)");
                printf("\n 9.Traverse the list (Right to Left) \n");
                printf("\n 10. Count nodes ");
```

```
printf("\n 11. Exit ");
printf("\n\n Enter your choice: ");
scanf("%d",&ch);
switch(ch)
{
        case 1:
                if(start == NULL)
                {
                        printf("\n Number of nodes you want to create: ");
                        scanf("%d", &n);
                        createlist(n);
                        printf("\n List created..");
                }
                else
                        printf("\n List is already created..");
                break;
        case 2:
                insert_at_beg();
                break;
        case 3:
                insert_at_end();
                break;
        case 4:
                insert_at_mid();
                break;
        case 5:
                delete_at_beg();
                break;
        case 6:
                delete_at_last();
                break;
        case 7:
                delete_at_mid();
                break;
        case 8:
                printf("\n The contents of List (Left to Right): \n");
                traverse();
                break;
```

```
                    case 9:
                        printf("\n The contents of List (Right to Left): \n");
                        rev_traverse(start);
                        break;
                    case 10:
                        printf("\n No of nodes : %d ", countnode(start));
                        break;
                    case 11:
                        exit(0);
                }
                getch();
        }
}


node* getnode()
{
    node * newnode;
    newnode = (node *) malloc(sizeof(node));
    printf("\n Enter data: ");
    scanf("%d", &newnode -> data);
    newnode -> next = NULL;
    return newnode;
}



int countnode(node *ptr)
{
    int count=0;
    while(ptr != NULL)
    {
       count++;
       ptr = ptr -> next;
    }
    return (count);
}
```

```
void createlist(int n)
{
    int i;
    node *newnode;
    node *temp;
    for(i = 0; i < n; i++)
    {
       newnode = getnode();
        if(start == NULL)
        {
                start = newnode;

        }
       else
        {
                emp = start;
                while(temp -> next != NULL)
                        temp = temp -> next;
                temp -> next = newnode;
       }
    }
}


void traverse()
{
    node *temp;
    temp = start;
    printf("\n The contents of List (Left to Right): \n");
    if(start == NULL)
    {
       printf("\n Empty List");
       return;
     }
    else
    {
       while(temp != NULL)
        {
                printf("%d-->", temp -> data);
```

```
                temp = temp -> next;
            }
        }
}


void rev_traverse(node *start)
{
    if(start == NULL)
    {
                return;
    }
    else
    {
                rev_traverse(start -> next);
                printf("%d -->", start -> data);
    }
}


void insert_at_beg()
{
    node *newnode;
    newnode = getnode();
        if(start == NULL)
        {
                start =newnode;
        }
        else
        {
                newnode -> next = start;
                start =newnode;
        }
}

void insert_at_end()
{
    node *newnode, *temp;
    newnode = getnode();
    if(start == NULL)
```

```
    {
       start = newnode;
    }
    else
    {
       temp = start;
       while(temp -> next != NULL)
              temp = temp -> next;
       temp -> next = newnode;
     }
}

void insert_at_mid()
{
    node *newnode, *temp, *prev;
    int pos, nodectr, ctr = 1;
    newnode = getnode();
    printf("\n Enter the position: ");
    scanf("%d",&pos);
    nodectr = countnode(start);
    if(pos > 1 && pos < nodectr)
    {
       temp = prev = start;
       while(ctr < pos)
        {
              prev = temp;
              emp = temp -> next;
              ctr++;
       }
       prev -> next = newnode;
       newnode -> next = temp;
    }
    else
       printf("position %d is not a middle position", pos);
}
```

```
void delete_at_beg()
{
    node *temp;
    if(start == NULL)
    {
        printf("\n No nodes are exist..");
        return ;
    }
    else
    {
        temp = start;
        start = temp -> next;
        free(temp);
        printf("\n Node deleted ");
    }
}

void delete_at_last()
{
    node *temp, *prev;
    if(start == NULL)
    {
        printf("\n Empty List..");
        return ;
    }
    else
    {
        emp = start;
        prev = start;
        while(temp -> next != NULL)
        {
                prev = temp;
                temp = temp -> next;
        }
        prev -> next = NULL;
        free(temp);
        printf("\n Node deleted ");
    }
}
```

```
void delete_at_mid()
{
	int ctr = 1, pos, nodectr;
	node *temp, *prev;
	if(start == NULL)
	{
		printf("\n Empty List..");
		return ;
	}
	else
	{
		printf("\n Enter position of node to delete: ");
		scanf("%d", &pos);
		nodectr = countnode(start);
		if(pos > nodectr)
		{
			printf("\nThis node doesnot exist");
		}
		if(pos > 1 && pos < nodectr)
		{
			temp = prev = start;
			while(ctr < pos)
			{
				prev = temp;
				temp = temp -> next;
				ctr ++;
			}
			prev -> next = temp -> next;
			free(temp);
			printf("\n Node deleted..");
		}
		else
		{
			printf("\n Invalid position..");
			getch();
		}
	}
}
```